情報処理学会論文誌 Vol.58 No.4 871–884 (Apr. 2017)

ソフトウェア開発PBLにおけるビルドエラーの調査

槇原 絵里奈[1,a)] 井垣 宏[2] 吉田 則裕[3] 藤原 賢二[4] 川島 尚己[1,†1] 飯田 元[1]

受付日 2016年8月10日, 採録日 2017年1月10日

概要：ソフトウェア開発においてビルドは重要な工程の 1 つである．多くの既存研究が企業のソフトウェア開発においてビルドが成果物の品質へ与える影響やビルドの支援手法を調査している．一方，高等教育機関で開講されている，ソフトウェア開発 PBL 演習における学生のビルド活動の実態調査は十分に行われていない．そこで，我々はアジャイル開発を取り入れたソフトウェア開発 PBL において，学生のビルドエラーの実態調査を行い，学生が陥りやすいビルドエラーの特徴を分析した．具体的には，学生が個人の開発環境において行うローカルビルドと，チーム共用開発環境で行われるリモートビルドを調査した．ローカル/リモートビルドにおけるエラーの種類，各種エラーの回数や解決時間を調査・比較したところ，学生も実務家同様ファイル間の要素の依存関係によるエラーが多いことや，ローカルでビルドを頻繁に実行・結果を確認することでリモートでのエラー発生を防ぐことが可能なエラー種類などが判明した．これらの結果は，ソフトウェア開発 PBL において教員が優先的に指導・確認すべき項目として扱うことが可能である．

キーワード：ビルド，PBL，ソフトウェア開発演習，アジャイル開発，ソフトウェア工学教育

Investigation of Build Error on Software Development PBL

Erina Makihara[1,a)] Hiroshi Igaki[2] Norihiro Yoshida[3]
Kenji Fujiwara[4] Naoki Kawashima[1,†1] Hajimu Iida[1]

Received: August 10, 2016, Accepted: January 10, 2017

Abstract: Software build is a fundamental process in software development. Many previous research have investigated the impact of build on the quality of software product in industrial software development. In addition, they also proposed supporting tool and method for the build process. However, there is no existing research focusing on investigating the build activities performed by students in college software development PBL practice course. Therefore, in our research, we investigated and analyzed students' build errors occurring in such Project-Based Learning (PBL) practice course, which adopted agile development. We collected and analyzed two main sets of data, as one is the local build log from each student's personal development environment, and another is the remote build log from each team's common development environment. Based on the collected data, we compared the build error types, the count of each error type and their respective fixed time between the local and the remote build. From our results, we confirmed that not only experts but also students experience most errors which come from the dependency between files' elements. Furthermore, we identified the error types which students can prevent by frequently performing local build and confirming the results before they submit for remote build. These results can indicate which particular factors that educators should refer to with priority and provide students with corresponding supports and assists.

Keywords: software build, PBL, software development practice, agile, software engineering education

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[2] 大阪工業大学
Osaka Institute of Technology, Hirakata, Osaka 573–0196, Japan
[3] 名古屋大学
Nagoya University, Nagoya, Aichi 464–8601, Japan
[4] 国立高等専門学校機構豊田工業高等専門学校
National Institute of Technology, Toyota College, Toyota, Aichi 471–8525, Japan
[†1] 現在，日立ソリューションズ
Presently with Hitachi Solutions, Ltd.
[a)] makihara.erina.lx0@is.naist.jp

1. はじめに

近年のソフトウェア需要拡大・産業連携を背景に，IT 教育の分野では実践的なソフトウェア開発能力を有する人材育成が注目されている [1]．ここで述べる実践的なソフトウェア開発能力とは，プログラミングの知識だけではなく，多様な人材と交流しそれらをマネジメントする力，コミュニケーション能力，課題を発見・提案する能力などを含む．そこで，近年情報系学部・学科を有する高等教育機関において特定のアプリケーションやシステムをチームで開発す

るソフトウェア開発教育が重要視されている．特に，複数人の学生がチームを組みプロジェクト形式で実施されるソフトウェア開発演習（ソフトウェア開発 PBL [2], [3]）がさかんになりつつあり，最近では従来のウォーターフォール型の開発だけでなく，アジャイル開発を取り入れたソフトウェア開発 PBL も行われている [4], [5], [6]．アジャイル開発では，ウォーターフォール型開発と比較し，実装，テスト，ビルド，デプロイといった工程を短い期間で継続的に何回も繰り返すことが求められる．アジャイル開発を取り入れたソフトウェア開発 PBL においても，正常に動作するソフトウェアを継続的に構築するため，実装やテストだけでなく，ビルドやデプロイといった工程の重要性を学生に教育することが必要となる．ここでビルドとは，ソースコードのコンパイルやライブラリのリンクなどを行い，最終的な実行可能ファイルを作成する工程を指し [7]，デプロイとはシステムをサーバへアップロードし，他のユーザやプログラムがシステムを利用可能な状態に展開する工程である [8]．多くの既存研究がビルドはソフトウェア開発に欠かせない工程であると述べており，ビルドの工程や結果がソフトウェア品質へもたらす影響 [9], [10], [11] やビルドの支援ツールに関する様々な研究 [12], [13], [14], [15] が行われている．Microsoft や Google など企業でもビルドおよびビルドエラーに関する分析がなされており [10], [12], [16]，ビルドエラーが生じる原因を明らかにする取り組みが行われている．一方でソフトウェア開発 PBL をはじめとしたチームを対象としたソフトウェア開発教育において，ビルド工程に関する調査はほとんど行われていない．

そこで，我々は関西圏の修士 1 年生を対象としたソフトウェア開発 PBL である CloudSpiral におけるビルド活動の分析を行った．具体的には，2013 年度から 2015 年度にかけて，CloudSpiral の演習科目として行われたウェブアプリケーション開発におけるローカル/リモートのビルドログを，以下のリサーチクエスチョンに沿って分析した．

RQ1 どのような種類のビルドエラーが存在するか．

RQ2 どれぐらいの頻度で各種ビルドエラーは生じるか．

RQ3 各種ビルドエラーの解決にはどれぐらいの時間がかかるか．

我々はこれらのリサーチクエスチョンを通し，PBL のようなチーム開発におけるローカル/リモートビルドの難所を調査する．本研究で得られた結果は，教員がソフトウェア開発 PBL において学生をどのように指導・補助すべきかを考慮する重要な観点として活用することができると考える．

本論文の構成を以下に示す．まず 2 章で本研究が対象とするソフトウェア開発 PBL の説明を行い，次にソフトウェア開発におけるビルドや，ソフトウェア開発教育におけるビルド・コンパイルに関する分析を行っている研究とその課題を述べる．3 章では我々が実際に分析を行った CloudSpiral の分析データについて詳述する．4 章では各リサーチクエスチョンの結果および考察を述べる．そして 6 章ではリサーチクエスチョンの結果全体に対する考察を述べた後，7 章をまとめとする．

2. 準備

本章では本研究に関連する技術，既存研究について説明する．まず 2.1 節では，ソフトウェア開発 PBL および実際に分析対象とした CloudSpiral について説明する．次に 2.2 節でソフトウェア開発プロセスにおけるビルドについて関連研究を交え説明した後，2.3 節においてソフトウェア工学教育におけるビルドおよびビルド工程の一部にあたるコンパイルの分析を行っている既存研究について述べる．

2.1 ソフトウェア開発 PBL

Project-Based Learning（以下 PBL と呼ぶ）とは実際のプロジェクトを通して学生の課題発見・課題解決を実現する科目であり，近年教育機関において注目されている授業形態の 1 つである [1]．PBL を通して参加者は，参加者同士のコミュニケーションだけではなく，自分で課題を発見する力，学生間の能力の差を解決するための力などが養われると考えられる．

実際に教育機関において開講されているソフトウェア開発 PBL として CloudSpiral があげられる．CloudSpiral[*1] とは分野・地域を超えた実践的情報教育協同ネットワーク（Education Network for Practical Information Technologies，略称 enPiT）[*2] の取り組みの 1 つとして，大阪大学と神戸大学を中心に 2013 年度から行われている教育プログラムである [17]．主な対象は関西の大学院に所属する修士 1 年生であり，学生はそれぞれ 5，6 人のチームに分けられ，1 年間の座学・演習と複数回のソフトウェア開発 PBL を通してアジャイル開発手法やクラウドコンピューティングの基礎から応用を学習する．具体的な座学・演習の内容として，プロジェクトを円滑に進めるためのファシリテーションスキルや，LEGO を用いた Scrum・チケット駆動開発演習などがある．さらに，座学・演習の内容をふまえ，1〜3 週間程度の期間でアジャイル開発手法に基づいて，チームでソフトウェアを開発するソフトウェア開発 PBL が複数回実施されている．

他にもアジャイル開発を取り入れたソフトウェア開発 PBL は様々な教育機関で実施されている．国内では，産業技術大学院大学[*3] が革新的な教育の枠組み[*4] として，ビジネスアプリケーションなどソフトウェア・システム開発において PBL を利用している．九州大学[*5] でも，修士を対

[*1] http://cloud-spiral.enpit.jp/
[*2] http://www.enpit.jp/
[*3] http://aiit.ac.jp/
[*4] http://aiit.ac.jp/education/
[*5] http://www.qito.kyushu-u.ac.jp/

象に情報通信技術に関する高い実践力を持つ学生の育成を試みている．国外でも，dos Santos ら [5] はブラジル内の実企業のインターンシップにおける PBL 評価モデルを提唱しており，Kropp ら [6] は，チューリッヒ大学で開講されているアジャイル形式のソフトウェア開発演習を実施・改善している．

2.2 ソフトウェア開発プロセスにおけるビルド

ビルドとは，ソースコードのコンパイルやライブラリのリンクなどを行い，最終的な実行可能ファイルを作成する工程を指す [7]．多くの既存研究がビルドはソフトウェア開発に欠かせない工程であると述べている [9], [10], [16]．一方，Kerzazi ら [18] は，ビルドは重要な工程であるにもかかわらず，既存の研究はビルドが失敗する原因や，失敗したことによる影響の調査が不十分であると指摘している．彼らはビルドエラーが生じる原因の 1 つに，役割が違う開発者間でのコミュニケーション不足があると報告している．同様に，Phillips ら [11] も，ビルドチームの課題は技術的な要素ではなく技術者間のコミュニケーションや知識共有など，非技術的なものであると述べている [19]．Seo ら [16] は Google の開発履歴を基に，ビルドエラーの分類やどの種類のエラーが頻出するかなどについて調査を行った．彼らの調査の結果，必要なパッケージや変数の型名が見つからなかったときに生じる Dependency（依存関係の不整合）のエラーが最も多いことが明らかとなった．

これらの調査は Microsoft や Google など企業において行われていることが多い [10], [12], [16]．既存研究によると熟練の開発者でもビルドエラーの修正には大幅なコストがかかるとされている [18]．

以上の既存研究より，ソフトウェア開発プロジェクトにおいてビルドは重要な工程であり，ビルド時に発生するエラーには技術的な要素だけでなく，チームメンバ間のコミュニケーションや知識共有といった人的な要素にも起因することが分かっている．そのため，学生がどのようなビルドエラーを起こしやすいのか，またその原因や解決にどれだけのエフォートを必要とするのかを明らかにすることは，ソフトウェア開発教育において非常に重要な課題である．次節では，ソフトウェア開発教育において行われているビルドやコンパイル時の学生の振舞いに関する既存研究について説明する．

2.3 ソフトウェア工学教育におけるコンパイル・ビルド分析

本節では初学者によるプログラミングを行う際の一連の行動と成果物，たとえば，ソースコードのスナップショット，コンパイルや実行時の結果など（以下プログラミング行動 [20] と呼ぶ）のログを分析し教育へ応用した既存研究について述べる．

Jadud [21] は BlueJ [*6] を用いて初学者のコンパイル時のプログラミング行動のログを収集・分析した．彼らの調査の結果，初学者はシンタックスエラーを解決するために何回も修正・コンパイルを繰り返すことが確認された．Denny ら [22] は CodeWrite [*7] という，プログラミングにおけるテストの支援を目的としたウェブアプリケーションを開発し，データを収集した．彼らは収集したデータからシンタックスエラーの解決時間と成績の関係性について調査を行った．調査では，セミコロン抜けを修正する時間は成績が良い学生のほうが早かったものの，変数・メソッドなどの型の不一致や，識別子に関するエラーの解決時間は学生の成績とほぼ無関係であったと報告している．Hartmann ら [23] はコンパイルエラー時の修正案を提示する HelpMeOut というウェブアプリケーションを開発した．提示される修正案は，事前に他の学生が同じ課題を解いた際のソースコードが基になっている．コンパイルの成否に関係なく，HelpMeOut 上でコーディングされたソースコードおよび結果は，データベースへすべて保存され，どのような修正でどのようなエラーが生じたのかを閲覧できるようになっている．

これらの既存研究では，初学者のプログラミング行動から初学者がコンパイルやテストを行うときの難所・問題点を調査し，解決策を講じている．このような初学者のプログラミング行動の調査は個人開発を対象としている場合が多く [20], [24], [25], [26], [27], [28]，ソフトウェア開発 PBL のようなチームでのソフトウェア開発における，実装後のテストやビルドといった工程での学生の振舞いについてはまだ報告されていない．そこで我々は 2.1 節で先述した CloudSpiral において実施されたソフトウェア開発 PBL における学生チームによるビルドに着目し，調査を行う．

3. 調査対象

本研究では 2.1 節で説明した CloudSpiral の年間カリキュラムのうち，アジャイル開発手法に基づいてチームでウェブアプリケーション開発を主に行う，クラウド基礎 PBL 科目におけるビルド活動に着目し，調査する．学生はクラウド基礎 PBL が行われる前に，開発に必要なプログラミングの基礎技術，設計書や仕様書の読み方，開発環境や使用ツールの操作方法，レビューやテストの手法についてひととおり受講済みである．以下，クラウド基礎 PBL について詳述する．

クラウド基礎 PBL は 8 月に 5 日間の短期集中合宿として実施されるソフトウェア開発 PBL である．なお，開発を主に行うのは合宿最終日を除く 4 日間（それぞれ Sprint1 から 4 と呼称する）で，最終日は成果物や開発における工夫点の報告会が行われる．また，合宿が行われる前に，開発

[*6] http://www.bluej.org
[*7] http://codewrite.cs.auckland.ac.nz/

表 1 チーム概要
Table 1 Statistics of teams for each year.

開講年度（年）	2013	2014	2015
参加人数（人）	49	54	46
チーム数	9	9	8
ログ収集期間	7 月 26 日 ～8 月 22 日	7 月 25 日 ～8 月 21 日	7 月 24 日 ～8 月 20 日

の流れを理解するための練習日（preSprint に対応）が用意されており，学生は preSprint から Sprint4 までの期間，与えられた仕様書と開発環境を利用して，Java，JavaScript，mongodb といった技術を利用してウェブアプリケーションを開発する．開発対象はチケット販売システムと呼ばれるウェブアプリケーションであり，ログイン，アカウント作成，チケット購入といった 13 種類の機能について仕様書が与えられる．仕様書が共通なため，各チームによって開発されるものも共通であるが，チケット販売システムのどの機能をどういった順序で開発するか，開発中のメンバの作業分担といった開発プロセスの詳細については学生らに任されている．なお，本調査対象のプロジェクト終了時点における平均行数は約 6,000 行，ファイル数は約 100 ファイルに及ぶ．

また，各機能においてユーザの入力を検証するための仕組みとして Bean Validation と呼ばれるバリデータを用いる．バリデータの作成において，以下の枠で囲んだソースコードのように Java 言語の Annotation 機能を利用して行う．Annotation については，各機能の全テストにおいても利用される．

```
10: @Length(min=4, max=12,
      message="パスワードには 4 から 12 文字の
                英数字のみが利用できます")
11: @Pattern(regexp="^[a-zA-Z0-9]*$",
      message="パスワードには 4 から 12 文字の
                英数字のみが利用できます")
12: private String pass;
```

本研究ではこのウェブアプリケーションを開発する際のビルドログを分析する．対象期間は 2013 年から 2015 年の 3 年間である．各年度の概要を**表 1** に，開発の流れを**図 1** に示す．以下，開発環境および開発の流れ，そして今回調査したログ種別について説明する．

3.1 開発環境

各学生は国立情報学研究所で運用されているサーバ上の個人用 Virtual Machine（以下 VM と呼ぶ）と大阪大学のサーバ上で運用されているチーム用 VM を利用し，ウェブアプリケーションの開発を行う．個人用 VM には Windows がインストールされており，開発環境として JDK，eclipse

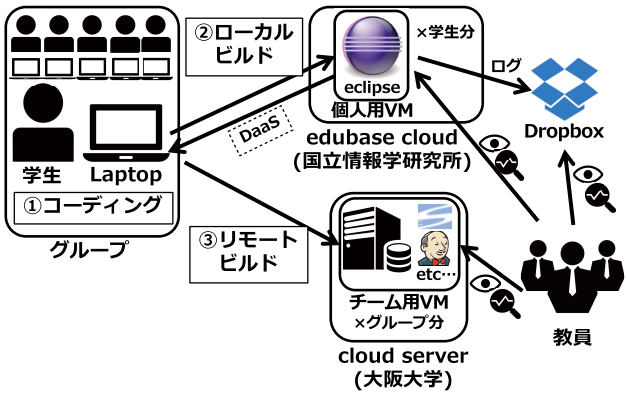


図 1 CloudSpiral における開発の流れ
Fig. 1 Development flow in CloudSpiral.

（ビルドツールの Apache Ant 含む各種プラグイン導入済み）[*8]，Apache Tomcat（アプリケーションサーバ）などが教員によって導入されている．学生は 1 人 1 つの個人用 VM にアクセスし，開発を行うことができる．

チーム用 VM には，Linux の Distribution の 1 つである CentOS がインストールされており，版管理システムである Subversion [*9]（以下 SVN と呼ぶ）や Ant [*10]，Tomcat [*11]，サーバでのビルドを支援する Jenkins [*12] など，開発に必要な様々なツールが事前に教員によって準備されている．学生はチームで 1 つのチーム用 VM を利用し，ソースコードの共有や完成したウェブアプリケーションの動作確認などを行うことができる．

これらの開発環境を学生がどのような手順で利用するかを以下に述べる．

手順 1：環境へのログインおよび実装

各学生は個人用 VM へとログインする．個人用 VM で eclipse を起動し，仕様書およびチームで決定したタスク分担に従い，各種ソースコード（テストコードや設定ファイルなど含む）の実装を行う．

手順 2：ローカルビルドおよび SVN へのコミット

チームでソフトウェア開発を行う場合，実社会におけるチーム開発では版管理システムでソースコードを共有することが多い[*13]．そのため，本 PBL においても学生が手順 1 における実装終了後に他のメンバとソースコードなどを共有するため，版管理システム（SVN）へのコミットを行う．このとき，もし学生が実装した部分に誤りがあった場合，誤りを含んだソースコードが SVN リポジトリにコミットされてしまう．本 PBL では，各学生が SVN リポジトリのチェックアウトを行うタイミングや回数は特に制限を設けておらず，いずれかのチームメンバが SVN リポジトリ

[*8] https://eclipse.org/
[*9] https://subversion.apache.org/
[*10] http://ant.apache.org/
[*11] http://tomcat.apache.org/
[*12] https://jenkins.io/
[*13] https://source.android.com/source/life-of-a-patch.html

へ対しコミットを行ったら，そのつどチェックアウトを行い自身のローカルの環境を最新版へ更新するように指導していた．したがって，SVN リポジトリへエラーを含んだソースコードをコミットすることは，結果として他のチームメンバへ，自身がソースコードへ加えたエラーを拡散させてしまうことにつながる．このような現象を本論文ではエラーの伝播と定義する．

エラーの伝播を起こさないためにも，本 PBL では実装が完了したことを確認するために，コミットを行う前に個人用 VM においてビルドを行うことを推奨し，事前学習においてその重要性についての教育を行っている．ここでは，個人用 VM において，学生自身が Apache Ant と呼ばれるツールを用いて実施するビルド作業のことをローカルビルドと呼ぶ．本 PBL におけるローカルビルドでは，コンパイル・テスト・コンパイル済みファイルやライブラリのリンクおよびアーカイブ・アプリケーションサーバへの配備，といった一連のプロセスが実施される．そのため，理想的な流れとしては，手順 1 で実装を行った後にローカルビルドを行い，ビルドが正常に実行されることやデプロイによってウェブアプリケーションが正常に動作することを各学生が各自の環境で確認した後に，SVN にコミットすることになる．ここでもしローカルビルドが失敗した場合，再度手順 1 へ戻り，誤りの修正を行った後に SVN へコミットする．

手順 3：リモートビルド

手順 2 が完了し，SVN にコミットが行われると，実装の完了した正常に動作するソースコードがチーム内で共有されることになる．コミットされたソースコードに問題がなければ，その後のタスク分担に従い，他の学生が手順 1 から開発を継続することになる．一方で，手順 2 でローカルビルドを怠ったままコミットが行われていたり，ローカルビルドが正しく実施されていても，他の学生によるコミットとの兼ね合いで，誤りを含んだソースコードが SVN リポジトリにコミットされてしまうことがある．そこでアジャイル開発を取り入れたソフトウェア開発プロジェクトの多くでは，本 PBL でも利用する Jenkins のようなツールを利用し，版管理システムにコミットされた内容を取得し，ビルドが正常に実行できるかを検証する仕組みを導入している．本 PBL においても，学生によるコミットが行われると，チーム用 VM に導入された Jenkins が自動的に SVN にアクセスし，コミットされた内容を取得し，ビルドを行い，結果を記録する．このように，学生のリポジトリへのコミットに連動して，Jenkins がチーム用 VM 上で実施するビルド作業のことをリモートビルドと呼ぶ．なお，リモートビルドとして行われるプロセスの内容はローカルビルドと同様である．学生は Jenkins によるリモートビルドの結果を確認し，成功したら次のタスクへとりかかり，失敗したら手順 1 へ戻りエラーの修正に注力することになる．

3.2 開発ルール

一般に，ソフトウェア開発プロジェクトを進めるにあたって，開発チームは様々なルールを守らなければならない[*14]．たとえば，原則としてチームによって開発されたすべてのソースコードは適切にレビューされなければならない，といったルールが実際に守られている．クラウド基礎 PBL においても，一般のソフトウェア開発プロジェクトに倣い，複数のルールが学生らのチームに提示されている．以下に教員側から定めたルールについて本論文に関係のあるものを抜粋して提示する．

- Sprint1 から 4 の開発可能時間は原則 10 時から 17 時のみで，それ以外の時間に個人用 VM での実装やレビューといった開発作業を行ってはならない（ただし，振り返りや次 Sprint の準備は認められている）．ただし，準備期間である preSprint の開発時間は指定された日の 13 時から 18 時までおよび Sprint1 が開始される直前までとする．
- SVN リポジトリにコミットを行う際，ビルドが失敗せず，原則としてその時点で存在するすべてのテストケースが正常に動作していることを確認しなければならない．
- チームメンバ間で，ソースコードおよびテストコードの実装回数，レビューの回数がなるべく均等に[*15]なるように，タスクを分配すること．このルールはソフトウェア開発 PBL でありがちな，特定の学生のみに実装・テストなどの負荷が集中し，経験の偏りが生じるのを防ぐことを目的としている [29].

3.3 調査対象ログ

本論文で調査するのは，ローカルビルド，リモートビルドおよび SVN リポジトリの各ログである．ローカルビルドは学生の個人用 VM で Apache Ant を利用して行われる．そのため，ローカルビルドは Ant によりビルド定義ファイルが読み込まれてから，ビルドが終了するまでを 1 回のローカルビルドとし，その間に記録されるビルド開始・終了時刻，ビルド成功/失敗，ビルド失敗時のエラーメッセージ，ローカルビルド実施学生，ビルド実施内容詳細をまとめて，そのローカルビルドのビルドログとして調査した．

リモートビルドは学生がチーム用 VM の SVN にコミットを行った際に，Jenkins が Ant を利用して自動的に行うものである．そのため学生がコミットを行い，Jenkins が連動して SVN から更新内容を取得し始めたときから，リモートビルドが終了して成功/失敗が Jenkins によって表示されるまでを 1 回のリモートビルドとする．リモートビルドのビルドログには，ローカルビルドと同様のビルド開

[*14] https://source.android.com/source/life-of-a-patch.html
[*15] 各メンバのファイル単位での実装・レビュー回数がチームの平均の 0.75〜1.25 倍以内に収まること

始・終了時刻，ビルド成功/失敗，ビルド失敗時のエラーメッセージ，ビルド実施内容詳細と SVN リポジトリに対して行われたどのコミットに連動してリモートビルドが実施されたかといった情報が含まれる．

SVN リポジトリには学生のすべてのコミットが記録されており，誰がいつどのようなコミットを行ったかを確認することができる．本論文では，ローカル/リモートビルドにおいて失敗が記録されたとき，ビルドエラーが発生したものとする．リモートビルドにおいて，どのようなコミットが行われたときにビルドが失敗し，エラーが起きたのかを調査する．

以降ではこれらのログを利用し，何を調査するかをリサーチクエスチョンとして示す．

4. リサーチクエスチョン

本研究の目的は，ソフトウェア開発 PBL における学生のビルドの特徴および難所を明らかにすることである．本研究で得られた結果は，学生がソフトウェア開発 PBL のようなチーム開発を行う際，教員が何を留意して指導を行うべきかの指針として活用できると考える．

本研究の目的を達成するために，我々は以下のリサーチクエスチョンを設定する．

RQ1 どのような種類のビルドエラーが存在するか．

RQ2 どれぐらいの頻度で各種ビルドエラーは生じるか．

RQ3 各種ビルドエラーの解決にはどれぐらいの時間がかかるか．

以下の節で各リサーチクエスチョンについて詳述する．

4.1 RQ1：どのような種類のビルドエラーが存在するか

まず，学生が陥りやすいビルドエラーの原因を調べるために，ビルドエラーの分類分けを行う．このリサーチクエスチョンでは既存研究 [16] で行われたビルドエラーの分類を参考に，我々が収集したすべてのビルドエラーの分類分けを行う．

以下に Seo ら [16] の作成したビルドエラーの分類を示す．

C1：Dependency

特定のソースコード間における，変数やクラス，メソッド，ライブラリなどの依存関係に起因するエラーを Dependency として分類する．具体的には，クラス間やファイル間において必要なシンボルやパッケージが存在しなかったときに生じるエラーを指す．例として “シンボルを見つけられません（`cannot find symbol`）” というエラーメッセージがあげられる．これは変数名やメソッド名，クラス名など，本来別ファイルや別箇所で宣言されているべき要素が見つからなかったときに生じるエラーである．

C2：Syntax

文法ミスが原因のエラーを Syntax として分類する．例として中括弧のとじ忘れの際に出力される “式の開始が不正です（`illegal start of expression`）” や，セミコロン抜けによる “';' がありません（`';' expected`）” というエラーメッセージがあげられる．

C3：Type Mismatch

変数や引数の型の違いによって生じるエラーメッセージを Type Mismatch として分類する．例として違う型の変数を比較した際に生じる “型 xxx と型 yyy は比較できません（`incomparable types: xxx and yyy`）” や，異なる型の変数が引数で与えられたときの “xxx は yyy に適用できません（`xxx cannot be applied to yyy`）” などがあげられる．

C4：Semantic

文法は正しいがビルド時にコンパイルエラーが生じるソースコードのエラー文を Semantic として分類する．たとえば，アクセス権のない変数・クラスの使用など修飾子に関するエラーや，実行されない文がソースコード中に含まれていることで生じるエラーが含まれる．エラーメッセージの例として “xxx はパッケージ外からはアクセスできません（`xxx is not public; cannot be accessed from outside package`）” や，“この文に制御が移ることはありません（`Statement not reached`）” などがあげられる．

以上が Seo らの論文に沿った分類である．

4.2 RQ2：どれぐらいの頻度で各種ビルドエラーは生じるか

RQ2 では，まず RQ1 で分類したローカル/リモートビルドエラーの分類ごとのエラー数を求める．調査対象のプロジェクトでは，学生はあらかじめ教員から，リモートでエラーを発生させないようにローカルでビルド結果の確認を済ませてからコミット（リモートビルド）を行うこと，と指導されていた．よって，どのようなエラーが頻出するかだけでなく，ローカルとリモートで分けてビルド発生の傾向を調査することで，ローカルで解決可能・不可能なエラーも調査できると考える．

次にチーム内における各学生のエラー数およびエラー種類内訳，エラー率を求める．チームごとに頻出するエラーや，エラー率の高いチームのエラーの種類を調査することで，教員が優先的に指導すべきエラーが明らかになると考える．

4.3 RQ3：各種ビルドエラーの解決にはどれぐらいの時間がかかるか

RQ3 では RQ1 のエラー種類ごとのエラーの解決時間を求める．エラー解決時間の定義は Seo らの定めたものを参考にして，エラーが生じたビルドのビルド終了時間から，次にエラーが解決した（エラーが生じなくなった）ビルドが開始した時間までとする．解決に長く時間がかかるエラーを調べることで，RQ2 同様教員が優先して指導したり，また講義資料などで補足したりすべきエラーの種類が明らかになると考える．

5. 結果

本章では 4 章で述べた各リサーチクエスチョンに対する結果と考察を述べる．

5.1 RQ1：どのような種類のビルドエラーが存在するか

調査対象である各年度のビルド数の総計とビルド成功・失敗回数の内訳を**表 2** に示す．表 2 のエラーについて，エラーの分類を行った結果を**表 3** に示す．

RQ1 の調査の結果，我々は新たにエラー分類の項目へ Annotation と Environment を加えた．Annotation とは Java プログラムのクラスやメソッド，パッケージへ対し付加情報を記入する機能を指す．Annotation はメソッドの直前に `@xxx` の形で書かれ，たとえば直後に記述するメソッドがスーパクラスのメソッドをオーバライドしていることを示す `@Override` や，テストメソッドであることを示す `@Test` などが存在する．

Seo らは Override の Annotation に関するエラーを Semantic として分類していた．しかし，3.1 節で述べたとおり，本研究で対象とした PBL では，多くの Annotation に関するエラーが生じていた．そこで，Annotation に関するエラーの特徴を調査するため，従来の分類である Semantic から分離させ，新たに Annotation の分類を設定した．なお，本研究における Annotation および Semantic のエラー数の合計値が，Seo らの Senmantic エラー数に対応しているため，対比させることが可能である．

また，Environment はチームで開発しているプロジェクト外が原因で発生するビルドエラーを示している．今回行ったビルドは，ソースコードのコンパイルだけでなく，テストの実行やテスト結果の表示，デプロイといった複数の処理を含んでいる．そのため，それらの処理を実行する際に必要な Ant などの外部ツールが正常に実行しないことによるビルドの失敗が特にローカルビルドにおいて多数発生した．そこで，Tomcat や Ant など開発ツールが原因で生じたビルドエラーをまとめて Environment と分類し，以下のとおり定義を行った．

C5：Annotation

今回の調査対象であるプロジェクトは Bean Validation[*16]を一部利用しており，誤った型の Annotation を使用しようとした際に生じるエラーを Annotation として分類する．エラーメッセージの例として，“`The annotation @NotEmpty is disallowed for this data type`” をあげる．これは，`@NotEmpy` は String 型にしか使用できないが，Date 型の変数に対し宣言しようとした際に生じるエラーメッセージである．

C6：Environment

調査対象のプロジェクトで使用しているツールに起因するエラーはすべて Environment として分類した．たとえば，Tomcat を起動せずに eclipse でビルドを行ったときに生じる “`java.net.ConnectException: Connection refused`” や，データベースエンジンの不具合によるエラー，テスト結果ファイルの出力に失敗したなどのエラーが含まれる．

本プロジェクトでは開発環境はほぼ教員が用意している．また，本論文の目的はソフトウェア開発 PBL におけるビルドエラーの特徴や学生にとっての難所を探し教育へ活かすことであり，開発環境に由来する Environment のビルドエラーは今回の我々の調査目的にはあてはまらないため以降結果から省くこととする．

表 2 調査対象データの概要
Table 2 Statistics of the data we analyzed.

	LOCAL			REMOTE		
開講年度	2013	2014	2015	2013	2014	2015
成功ビルド数	2,395	1,506	1,670	2,309	1,610	1,764
失敗ビルド数	340	142	297	149	219	94
合計	2,735	1,648	1,967	2,458	1,829	1,858

表 3 ビルドエラーの分類
Table 3 Statistics of build errors.

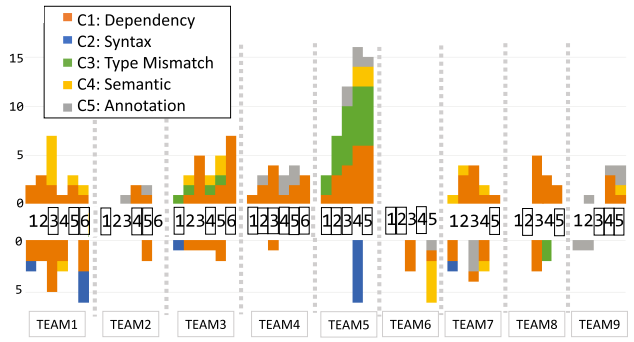
	LOCAL			REMOTE		
エラー分類	2013	2014	2015	2013	2014	2015
C1: Dependency	34	14	18	86	118	64
C2: Syntax	12	6	12	0	10	0
C3: Type Mismatch	2	3	1	28	27	18
C4: Semantic	6	16	2	19	5	10
C5: Annotation	6	22	11	16	59	2
(C6: Environment)	(280)	(81)	(253)	(0)	(0)	(0)
C1〜C5 の合計	60	61	44	149	219	94

5.2 RQ2：どれぐらいの頻度で各種ビルドエラーは生じるか

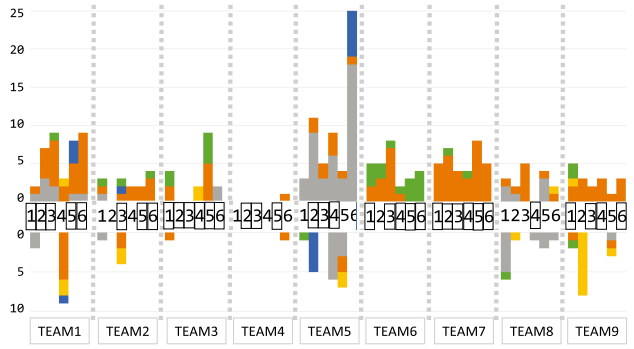
本節ではまず得られた結果の概要図・表について説明する．次にエラー各種の知見について述べる．

エラー分類ごとの件数は先述した表 3 のとおりである．チーム内における各学生のビルドエラーの分類のグラフを**図 2** に示す．横軸は各学生とチームに対応し，学生は灰色の破線でチームごとに区切られている．各学生には 1〜5 あるいは 6 までの ID が振られており，エラーの有無に関係なくローカルビルドよりリモートビルドのビルド回数が多かった学生の ID は枠線で囲まれている．すなわち，ID が枠線で囲まれている学生は，リモートビルドを行う前にローカルでビルドエラーが生じないか確認することを怠っていた可能性が高いことを示す．ローカルビルドよりリモートビルドが少ない・多い学生のエラー内訳につい

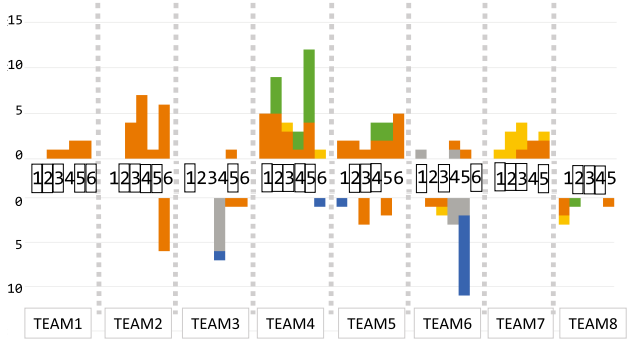
[*16] http://beanvalidation.org/



(a) 2013 年度



(b) 2014 年度



(c) 2015 年度

図 2 年度別チームごとのビルドエラー内訳（グラフ上部：リモートビルドのエラー内訳，グラフ下部：ローカルビルドのエラー内訳）

Fig. 2 Classification of each team's build error by year.

て，3 年分合計したものを**表 4** に詳細を示す．なお，ローカル/リモート両方のビルドエラー数が同じ学生は，ローカルのほうがビルド数が多い学生群へ分類し，0 の学生は表 4 へ含めていない．

また，ビルド数を比較するにあたって，ローカルビルドよりリモートビルドのほうが多い学生と少ない学生で人数が異なったため，エラー回数のみで比較することは困難であると考えた．そこで，エラー分類ごとのエラー数を，ローカルはローカルの 3 年間の総ビルド数で，リモートはリモートの 3 年間の総ビルド数で割った値に 100 をかけた値を E/T として表 4 へ記述した．すなわち，E/T の値は 3 年間のローカルビルド/リモートビルドにおける各エラーの占める割合にあたる．なお，3 年間のローカルビルド/リモートビルドの総数は表 2 から 3 の Environment を引いた値になり，ローカルビルドが 5,736 回，リモートビルドが 6,145 回である．

表 3 の調査の結果，全年度にわたってリモートビルドでは Dependency のエラーが最も多かった．これは Seo らの Google におけるビルドエラーの調査報告とも一致している．つまり，Dependency に関するビルドエラーは学生・実務家問わず発生しやすいエラーであることが分かる．さらに，ローカルビルドよりリモートビルドにおいて Dependency のエラーが多いことからも，Dependency はチーム開発における特徴的なビルドエラーであることが分かる．リモートにおける Dependency のエラーの中でも特に “シンボルを見つけられません（`cannot find symbol`）” というエラーが多く，リモートの Dependency エラーの 88%，リモート全体におけるエラーメッセージの 51%を占めていた．このエラーメッセージが生じる原因として，テスト対象のファイルが完成していないにもかかわらず，単体テストのテストファイル（以下，テストファイルと呼ぶ）をコミットしている事例を複数発見した．

たとえば，ユーザ登録機能を実現するためには，Account.java ファイルなどの複数のファイルを実装し，ファイルごとに AccountTest.java などのテストファイルを実装し，テストやレビューを実施する必要がある．ここで，Account.java とそのテストファイルにあたる AccountTest.java を別々に異なる学生が実装するようなケースにおいて，テストファイルを実装した学生がそのテスト対象をまだ実装・コミットしていないにもかかわらず，AccountTest.java のみをリポジトリにコミットしてしまうということが頻繁にあった．この場合，リポジトリには Account.java が存在しないため，リモートビルド時に，Account クラスが見つからない，つまりシンボルやパッケージが見つからないという Dependency エラーが生じてしまうことを防げない．このことから，Dependency エラーを防ぐためには，メンバ間のコミュニケーションが重要であるといえる．具体的にはまず各チームメンバが現在どのようなタスクを担当しているかを全員が把握している，あるいは即座に把握可能であるといった情報を共有することが必要であると考えられる．

3 章で述べたとおり，今回の調査対象プロジェクトにおいて学生らはチケット販売システムにおける 13 種類の機能（ユーザ登録機能，チケット販売機能，チケット登録機能など）の実装を行う．各機能は複数のファイルで構成されており，学生らは特定の機能を構成するすべてのファイルの実装，テスト，およびレビューを手分けして行う必要がある．このとき，どのファイルを誰がどのような順番で

表 4 ローカルビルドのほうが多い学生とリモートビルドのほうが多い学生の比較
Table 4 Comparison between students who tend to perform local and remote builds.

	ローカルのほうがビルド数が多い学生群 [56 名]				リモートのほうがビルド数が多い学生群 [92 名]			
	LOCAL		REMOTE		LOCAL		REMOTE	
エラー分類	エラー数	E/T(%)	エラー数	E/T(%)	エラー数	E/T(%)	エラー数	E/T(%)
C1: Dependency	44	0.77	79	1.29	22	0.38	189	3.08
C2: Syntax	22	0.38	6	0.10	8	0.14	4	0.07
C3: Type Mismatch	4	0.07	13	0.21	2	0.03	60	0.97
C4: Semantic	21	0.37	10	0.16	3	0.05	24	0.39
C5: Annotation	30	0.52	36	0.59	9	0.16	41	0.67

実装するかについては学生らが自由に決めることができる．単純な Syntax エラーのように，1 ファイル内の特定の箇所が原因で生じるエラーであれば，そのファイルを修正するのみで解決することが可能である．しかし，上記で述べたようなケースでは学生間で連携し，今からどのようなファイルを実装し，コミットするのか，そのコミットによってリポジトリの内容に不具合が生じないかどうかつねに注意を払う必要がある．上記事例では，チームメンバへ AccountTest.java の実装が完了し，コミットすることを周知させ，Account.java の実装・コミットが完了しているかを確認することが必要となる．

今回の調査対象プロジェクトでは，あらかじめ教員からリモートでビルドエラーを発生させないようにローカルでビルドを行ってからコミットを行うこと，と指導されていた．しかし，表 4 に示したように，実際には多くの学生がローカルビルドをあまりせずにリモートビルド（すなわちコミット）を行っていた．結果として，表 4 より，Syntax を除くすべてのエラー分類について，ローカルでビルドを頻繁に行っている学生群のほうがローカルビルドでのエラー率は高いが，リモートビルドにおけるエラー率は低くなっている．また，図 2 よりローカルビルドで Syntax エラーを確認した学生はリモートビルドで Syntax エラーを引き起こしておらず，かつリモートビルドで Syntax エラーを引き起こした学生は全員ローカルビルドで Syntax エラーを確認していないことが分かる．調査対象プロジェクトで用いた eclipse は基本機能として文法チェックや依存関係チェックを自動的に行うことが可能である．しかし，実際にローカル・リモートのビルドログを調査したところ，リモートビルドにおいて Syntax エラーが発生したすべての事例において，明らかに eclipse の文法チェックで検出されるような Syntax エラーであるにもかかわらずそれを無視し，ローカルビルドも実行せずに学生がリポジトリにコミットしていたことが判明した．

さらに詳細にエラー分類ごとのローカル/リモートのエラーを確認したところ，リモートビルドにおいて発生している Syntax エラー，Semantic エラー，Annotation エラーに分類されるエラーのなかに，学生が各自ローカルビルドを行っていれば解決可能であったエラーが複数存在することが判明した．我々の調査では，Syntax エラー，Semantic エラー，Annotation エラーに属するエラーには Syntax エラーに代表される，特定の単一ファイル内での記述が原因で発生するものと，本節でも述べたテストファイルとテスト対象を別々の学生が実装し，コミットした結果発生しているような複数ファイルにまたがる記述が原因で発生するものの 2 種類が存在する．この中で前者に分類されるエラーは，その原因が単一ファイルに閉じているため，ローカルビルドによってエラーを確認し，修正したのちに対象ファイルをコミットすれば，リモートビルドにおいて同一のエラーが発生することはない．以上より，ローカルビルドがリモートでのビルドエラーの削減に効果があり，教員はコミット前にローカルビルドを行うことの重要性を学生によりいっそう意識づける必要があると考えられる．

図 2 より，3 年間のうち Annotation のエラーは 2014 年度のチーム 5 で顕著に発生している．2014 年度のチーム 5 のビルドログを詳細に分析したところ，この Annotation エラーはすべて preSprint の日に生じており，preSprint 以降は 1 度も生じていなかった．このチーム以外にも，リモートで Annotation が生じているチームは 3 年間を通して 9 チームあるが，9 チーム中 7 チームは Anotation のエラーは preSprint から Sprint1 までの期間に生じており，残りの 2 チームも Sprint2 以降 Annotation のエラーは生じていない．つまり，Annotation のエラーは主に技術的な問題によるところが大きく，利用方法について習熟することで，ビルドエラーを削減できるといえる．

また，Dependency，Annotation に続き Type Mismatch に関するエラーがリモートで頻繁に生じていた．Type Mismatch のエラーが生じたソースコードを確認したところ，別のファイルでリストに格納する変数の型を宣言しているにもかかわらず，型の宣言を行っているファイルが完成していない状態でコミットを行っていることが判明した．

5.3 RQ3：各種ビルドエラーの解決にはどれぐらいの時間がかかるか

図 3 に過去 3 年分のローカルビルドとリモートビルドそ

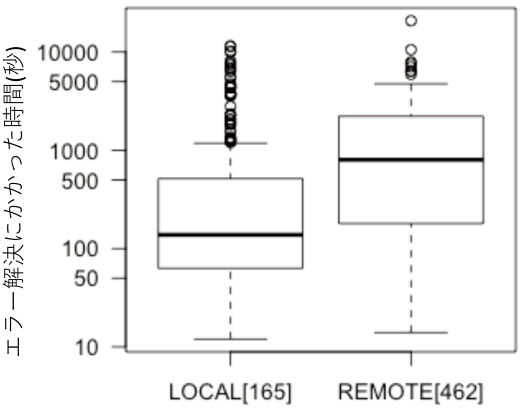


図 3 ローカル/リモートのエラー解決時間
Fig. 3 Error fixed time by local/remote.

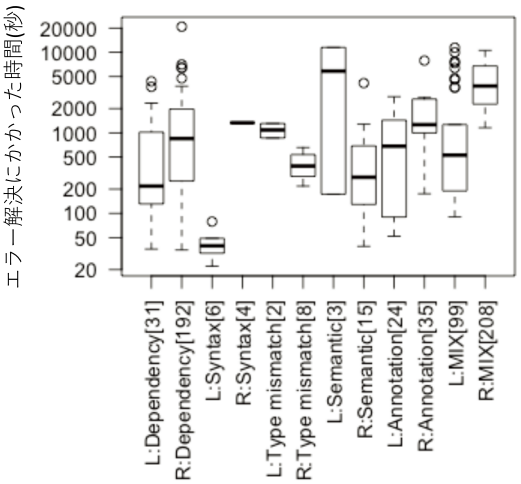


図 4 エラー分類ごとのエラー解決時間
Fig. 4 Error fixed time by each error type.

れぞれについてのエラー解決時間（秒）を示す．LOCAL，REMOTE の文字の横に書かれている数字はそれぞれのビルドエラー数を示している．エラー解決時間は 4.3 節で先述したとおり，特定のエラーが生じたビルドのビルド終了時間から，次にエラーが解決した（エラーが生じなくなった）ビルドの開始時刻までの時間である．箱ひげ図の中央値に着目したところ，全体的にリモートビルドのほうがエラー解決に時間がかかっていることが分かる．一方で，一部のローカルビルドにおけるエラーはリモートビルドにおけるエラーよりも解決に時間がかかっていることも見て取れる．

図 4 に過去 3 年分のエラー分類ごとのエラー解決時間（秒）を示す．エラー解決時間の定義は図 3 と同様であるが，エラーが生じてから，解決されるまでの間に発生したビルドエラーの種別ごとに分けてグラフを作成している．また，解決までに複数種類のビルドエラーが含まれていた場合は，MIX と定義して図 4 の一番右へ結果を示している．また，分類名横の括弧中の数値は，対象となったビルドエラー数にあたる．それぞれのエラー分類のうち，左がローカルでのエラー解決時間で，右がリモートでのエラー解決時間を示す．図 4 より，エラー数の少ない Syntax と Type mismatch，Semantic を除くと，残りの Dependency，Annotation，MIX はローカルよりリモートのほうがエラー解決に時間がかかっていることが分かる．さらに，リモートビルドのエラーの中でも MIX，すなわち，複数の要因によるビルドエラーが最も解決に時間がかかっていることが分かる．

6. 考察

本研究では RQ1 でリモートおよびローカルのビルドエラーの分類を行い，その結果をふまえて RQ2，RQ3 について調査を行った．本章では，RQ2，RQ3 の調査結果から得られた知見について考察を行う．

6.1 リモートビルドにおけるエラーの原因

RQ2 の結果をふまえ，リモートビルドにおけるエラーの原因について考察する．

リモートビルドにおいて発生するエラーには技術的な要因によるものと，学生間のコミュニケーションに関するものがあることが分かった．調査では，Syntax エラー，Semantic エラー，Annotation エラーのうち，単一ファイル内での記述が原因で発生しているエラーの多くは技術的な要因で発生していることが分かっている．たとえば，5.2 節で述べたとおり Semantic のエラーの一部や Syntax のエラーは，各個人がローカルでのビルドを怠っていたため，ローカルで確認可能なエラーがリモートにおいて生じていた．さらに，Annotation に関するエラーはすべて開発が始まってから 2 日以内に生じていた．このことから Annotation に関するエラーは学生が調査対象プロジェクトで使用していたバリデータの実装方法を理解していなかったため生じたと考えられる．

一方，Dependency や Type Mismatch はコミュニケーションに起因することが多いという結果が得られた．5.2 節より，リモートにおける Dependency のエラーはテスト対象のファイルがコミットされていないにもかかわらずテストファイルをコミットして生じることが多かった．さらに Type Mismatch で頻出していたエラーとして cant.apply.symbol があり，このエラーログを確認したところ Dependency の cant.resolve と同時に生じていることが多かった．これらのエラーが共起したコミットログを確認したところ，リストへ格納するデータの型を他のファイルで宣言しているにもかかわらず，宣言元のファイルが存在しない状態でコミットを行っている事例を確認した．

類似した事例として，Dependency の cant.resolve というエラーが生じるとき，テスト対象のファイルがコミットされていないにもかかわらず，テストファイルをコミットしている事例が確認された．これらは 5.2 節で述べたとお

り，チームメンバ間で誰がどのコンポーネントを担当しているかを確認し合うことで，エラーを防ぐことが可能である．以上より，Dependency や Type Mismatch に関するエラーは学生同士が口頭でコミュニケーションをとることで防ぐことが可能であったと考える．

発生したエラー数については，学生は複数ソースコード間におけるシンボルやパッケージの宣言ミス，すなわち Dependency（依存関係の不整合）によるエラーが最も多い．さらに Dependency によるエラーログを詳細に調べたところ，テストファイルのコミット時に Dependency のエラーが頻繁に生じており，コミットすべきファイルについての学生間のコミュニケーションに問題が発生していたといえる．この結果は Seo ら [16] や Phillips ら [11] といった企業での報告とも一致しており，ビルド工程における Dependency やコミュニケーションの問題はプログラミング能力の有無に関係なく発生するといえる．

6.2 ローカルビルドの重要性

RQ2 の結果をふまえ，ローカルビルドの重要性について考察する．

4.2 節で述べたとおり，本プロジェクトでは事前講義において，版管理システムを利用したチームソフトウェア開発におけるローカルビルドの重要性についての指導を行っている．しかし，5.2 節の表 4 で取り上げた，ローカルビルド数よりリモートビルド数が多い学生は，148 名のうち 92 名にのぼり全体の約 62%を占めた．教員の指導どおりに，リモートへコミット（リモートビルド）を行う前にローカルでソースコードの動作確認（ローカルビルド）を行っていた場合，リモートビルド数よりローカルビルド数が多くなるため，92 名については教員の指導どおりにビルドを行っていなかったことが分かる．

調査の結果においても，ローカルビルドを積極的に行っている学生のほうがリモートビルドでのエラー発生率が低くなっている．特に Syntax に関するビルドエラーのような技術的な要因によるエラーについて，学生がローカル環境でのビルドを行うことで見つけることが容易である．しかし，実際にはローカルビルドを行わない学生が，エラーが残ったままのソースコードを SVN リポジトリへコミットを行い，リモートビルドでエラーが生じてしまった事例が確認された．

一方で 2013 年度の Team2，3 のように，リモートビルドでのエラーが非常に少なく 3.1 節の手順 2 で定義したエラーの伝播もほとんど発生していないチームが各年度 2〜3 チーム存在する．これらのチームにヒアリングを行ったところ，早期段階でリモートビルドでのエラーが発生しないよう，メンバ間での意識共有を行っていたことが分かった．具体的には，ローカルビルドの徹底やコミット内容の事前確認，コミット時のセルフレビューといった様々な対策をチーム内で決定し，順守するよう相互に声を掛け合うといった対応をとっていた．結果として，これらの対策がリモートビルドにおけるエラーの発生やエラーの伝播を少なくしたと考えられる．

ソフトウェア開発 PBL を進めるうえでは，事前講義においてローカルビルドの重要性を伝えるだけでなく，学生チームが自発的に重要性を理解し，積極的にローカルビルドを実施するような働きかけを行うことが重要であると考えられる．

6.3 ローカル/リモートのビルドエラーの違い

RQ3 の結果をふまえ，ローカルとリモートビルドにおけるエラーの違いについて考察する．

版管理システムを用いたソフトウェア開発では，コミットを行う前にローカルビルドによってビルドエラーを洗い出しておき，リモートビルドでのビルドエラーを発生させないことが重要になる．ソフトウェア開発 PBL においても同様であり，調査において解決時間や解決を目的とした作業内容について，差異があることが調査により分かった．以降の段落において，解決時間および解決を目的として学生が行った作業内容に関する考察を述べる．

図 3 において，解決に 1 時間以上要しているローカルビルドにおけるエラーをより詳細に調査したところ，実装に不慣れな特定の学生が問題の特定や解決方法の探索に時間がかかり，悩んでいるケースが多いことが分かった．このようなケースでは，周囲の学生がフォローすることで，解決時間を早めることができる．そのためには，解決に時間がかかりそうだと分かった時点で，その学生自身がサポートを積極的に求めるといった対応をとらせることが重要である．

一方，リモートビルドにおいて，ビルドエラーの解決に 1 時間以上を要している 11 件を分析したところ，技術的な問題だけでなく，開発の進め方の問題で解決時間が長くなっている事例が見られた．たとえば，11 件中 7 件は前節で述べたような依存関係のあるファイル（テスト対象ファイル，リストに格納する要素の型を宣言しているファイル，インポート先のファイルなど）の実装が完了しコミットされるまで待っていたため，結果としてエラー解決時間が長引いていた．他にも，技術的に解決可能な学生の手が空いてなかった事例やバグが埋め込まれてからしばらくしてエラーが発生したため，即座に対応可能な学生がいなかった事例などを確認した．特に，2013 年度のチーム 5，2014 年度のチーム 1，7，9，2015 年度のチーム 4，5 では，学生の 1 人がリモートにおいてビルドエラーを発生させた後，解決を保留するなどした結果，他の学生がビルドエラーを発生させるソースコードを SVN からチェックアウトし，エラーの伝播が発生している．エラー解決時間については，図 4 からも分かるように，複数種類のエラーを含むこと

で，解決に時間を要してしまう．そのため，リモートビルドにおけるエラーについては，可及的速やかにエラー解決を行えるよう，開発の進め方をコントロールする方法を指導する必要があると考えられる．

6.4 PBL 教育への適用

本節では，RQ1 により得られた分類を基に，RQ2，RQ3 を調査した結果どのように PBL 教育へ適用可能かについて考察を述べる．

RQ2 より，技術的な要因に起因するエラーとコミュニケーション不足に起因するエラーの両者があることが分かった．技術的な要因により発生したエラーとして Syntax，Semantic，Annotation エラーがあげられる．これらは，単一ファイルに閉じたエラーであるため，ローカルビルドで発見が可能である．このことから，教員は演習前にこれらのエラーについて解決方法を提示し，ローカルビルドを徹底するよう指示必要があると考えられる．一方，チーム内のコミュニケーション不足に起因するエラーでは，エラーを引き起こすコードの特定や解決に時間がかかっていた．このようなエラーはテスト対象ファイルが完成していないにもかかわらず，テストファイルを SVN リポジトリへコミットするなど，複数ファイルに起因して生じることが多かった．RQ3 の調査結果より，エラーの修正を怠った結果，別のエラーが追加されてしまうと，エラー解決に時間がかかってしまうことが分かった．したがって，教員は学生に対して，依存関係のあるファイル（例：テストファイルや import 文を含むファイル）を SVN リポジトリへコミットするときは，エラーが生じないかメンバへ確認することを勧めるべきである．

さらに RQ2，RQ3 からローカルビルドを怠るとエラーの伝播が生じ，先述したように複数種類のエラーが混ざりエラー解決に時間を要することが判明した．よって，教員はローカルで生じたビルドエラーが，他のチームメンバへ伝搬する流れを学生へ説明し，演習前へローカルビルドでエラーを確認しなくすことの重要性を伝えるべきである．

7. おわりに

本研究ではソフトウェア開発におけるビルド工程に着目し，ソフトウェア開発 PBL における学生のビルド活動の調査を行った．調査の結果，我々は得られたビルドエラーを Seo らの分類を参考に，PBL におけるローカル環境，リモート環境のビルドログを 6 つの項目へ分類した．そして，それぞれの分類に対しエラーの回数，エラー継続時間などを測定し，ローカルとリモートでビルドの傾向を比べることで，ローカルで解決可能なエラーや，リモートで教員が優先して留意すべきエラーを調査した．調査の結果，既存研究において述べられているソースコード間のシンボルやパッケージや宣言といった Dependency（依存関係の不整合）によるエラーや，チーム内におけるコミュニケーション不足は，学生同士のチーム開発においても影響を及ぼすことが判明した．我々の分析結果は，教員が PBL 教育において優先して指導，確認すべき項目として扱うことが可能である．

今後の課題として，時系列によるエラーの変化の調査があげられる．PBL 初期段階/終了段階や Sprint 開始時/終了時などの，プロジェクトの段階に応じたエラーの発生傾向を調査することが可能である．これらの結果を組み合わせることで，教員は座学や振り返りを行うタイミングの選定，優先して指導すべきチームの早期発見ができると考える．

謝辞 本研究で調査対象とした CloudSpiral は，分野・地域を越えた実践的情報教育協働ネットワーク（通称 enPiT: Education Network for Practical Information Technologies）の一環として実施されました．enPiT の運営および，CloudSpiral の実施に関わった参画大学，関連企業の関係者，教員，学生の皆様へ深く感謝いたします．

参考文献

[1] 独立行政法人情報処理推進機構（IPA）IT 人材育成本部：IT 人材白書 2015 (2015).
[2] 眞鍋雄貴，井垣 宏，福安直樹，佐伯幸郎，楠本真二，井上克郎：細粒度プロジェクトモニタリングのための DaaS を利用したソフトウェア開発 PBL 支援環境の提案，電子情報通信学会技術研究報告，pp.73–78 (2012).
[3] 佐伯幸郎，井垣 宏，福安直樹，柗本真佑，楠本真二：ソフトウェア開発 PBL のための DaaS を利用した開発環境の構築，電子情報通信学会技術研究報告，pp.13–18 (2012).
[4] Missiroli, M., Russo, D. and Ciancarini, P.: Learning Agile Software Development in High School: An Investigation, *Proc. 38th International Conference on Software Engineering Companion*, pp.293–302 (2016).
[5] dos Santos, S.C. and Soaress, F.S.F.S.: Authentic Assessment in Software Engineering Education Based on PBL Principles: A Case Study in the Telecom Market, *Proc. 2013 International Conference on Software Engineering*, pp.1055–1062 (2013).
[6] Kropp, M. and Meier, A.: Teaching agile software development at university level: Values, management, and craftsmanship, *Proc. 2013 26th International Conference on Software Engineering Education and Training*, pp.179–188 (2013).
[7] Lee, K.A.: *The Buildmeister's Guide: Achieving Agile Software Delivery*, Lulu.com (2008).
[8] Humble, J. and Farley, D.: *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley Professional (2010).
[9] McIntosh, S., Adams, B., Nguyen, T.H., Kamei, Y. and Hassan, A.E.: An Empirical Study of Build Maintenance Effort, *Proc. 33rd International Conference on Software Engineering*, pp.141–150 (2011).
[10] Hassan, A.E. and Ken, Z.: Using Decision Trees to Predict the Certification Result of a Build, *Proc. 21st IEEE/ACM International Conference on Automated Software Engineering*, pp.189–198 (2006).
[11] Phillips, S., Zimmermann, T. and Bird, C.: Under-

standing and Improving Software Build Teams, *Proc. 36th International Conference on Software Engineering*, pp.735–744 (2014).
[12] Adams, B., Tromp, H., de Schutter, K. and de Meuter, W.: Design recovery and maintenance of build systems, *Proc. 23th International Conference on Software Maintenance*, pp.114–123 (2007).
[13] Macho, C., McIntosh, S. and Pinzger, M.: Predicting Build Co-changes with Source Code Change and Commit Categories, *Proc. 23th International Conference on Software Analysis, Evolution, and Reengineering*, pp.541–551 (2016).
[14] Neitsch, A., Wong, K. and Godfrey, M.W.: Build system issues in multilanguage software, *Proc. 28th IEEE International Conference on Software Maintenance*, pp.140–149 (2012).
[15] Shridhar, M., Adams, B. and Khomh, F.: A Qualitative Analysis of Software Build System Changes and Build Ownership Styles, *Proc. 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp.29:1–29:10 (2014).
[16] Seo, H., Sadowski, C., Elbaum, S., Aftandilian, E. and Bowdidge, R.: Programmers' Build Errors: A Case Study (at Google), *Proc. 36th International Conference on Software Engineering*, pp.724–734 (2014).
[17] 中村匡秀，井垣 宏，佐伯幸郎：Cloud Spiral の取り組み，日本ソフトウェア科学会大会論文集，Vol.30 (2013).
[18] Kerzazi, N., Khomh, F. and Adams, B.: Why Do Automated Builds Break? An Empirical Study, *Proc. 30th International Conference on Software Maintenance and Evolution*, pp.41–50 (2014).
[19] Adams, B., Bird, C., Khomh, F. and Moir, K.: 1st International Workshop on Release Engineering (RELENG 2013), *Proc. 2013 International Conference on Software Engineering* (2013).
[20] Vihavainen, A., Luukkainen, M. and Kurhila, J.: Using Students' Programming Behavior to Predict Success in an Introductory Mathematics Course, *Proc. 6th International Conference on Educational Data Mining*, pp.300–303 (2013).
[21] Jadud, M.C.: Methods and Tools for Exploring Novice Compilation Behaviour, *Proc. 2nd International Workshop on Computing Education Research*, pp.73–84 (2006).
[22] Denny, P., Luxton-Reilly, A. and Tempero, E.: All Syntax Errors Are Not Equal, *Proc. 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, pp.75–80 (2012).
[23] Hartmann, B., MacDougall, D., Brandt, J. and Klemmer, S.R.: What Would Other Programmers Do: Suggesting Solutions to Error Messages, *Proc. SIGCHI Conference on Human Factors in Computing Systems*, pp.1019–1028 (2010).
[24] 井垣 宏，齊藤 俊，井上亮文，中村亮太，楠本真二：プログラミング演習における進捗状況把握のためのコーディング過程可視化システム C3PV の提案，情報処理学会論文誌，Vol.54, No.1, pp.330–339 (2013).
[25] Fujiwara, K., Fushida, K., Tamada, H., Igaki, H. and Yoshida, N.: Why Novice Programmers Fall into a Pitfall?: Coding Pattern Analysis in Programming Exercise, *Proc. 2012 4th International Workshop on Empirical Software Engineering in Practice*, pp.46–51 (2012).
[26] Tamada, H., Ogino, A. and Ueda, H.: A Framework for Programming Process Measurement and Compile Error Interpretation for Novice Programmers, *Proc. Joint Conference of the 21th International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pp.233–238 (2011).
[27] 槇原絵里奈，藤原賢二，Uthayopas, P.，Chantrapornchai, C.，Fakcharoenphol, J.，井垣 宏，吉田則裕，飯田 元：日本とタイにおけるプログラミング初学者のプログラミング行動の比較，電子情報通信学会技術研究報告，Vol.114, No.260, pp.47–52 (2014).
[28] Spacco, J., Denny, P., Richards, B., Babcock, D., Hovemeyer, D., Moscola, J. and Duvall, R.: Analyzing Student Work Patterns Using Programming Exercise Data, *Proc. 46th ACM Technical Symposium on Computer Science Education*, pp.18–23 (2015).
[29] Igaki, H., Fukuyasu, N., Saiki, S., Matsumoto, S. and Kusumoto, S.: Quantitative Assessment with Using Ticket Driven Development for Teaching Scrum Framework, *Proc. 36th International Conference on Software Engineering*, pp.372–381 (2014).

槇原 絵里奈 （学生会員）

平成 25 年大阪工業大学情報科学部情報システム学科卒業．平成 27 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，同大学院博士後期課程に在学．修士（工学）．ソフトウェア工学教育，特にプログラミング教育支援に興味を持つ．

井垣 宏 （正会員）

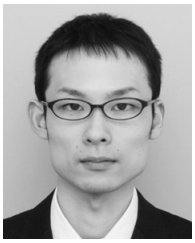
平成 12 年神戸大学工学部電気電子工学科卒業．平成 17 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年同大学院特任助手．平成 27 年大阪工業大学情報科学部准教授．博士（工学）．ソフトウェア工学教育，サービス指向アーキテクチャ，ソフトウェアプロセス等の研究に従事．

吉田 則裕 （正会員）

平成 16 年九州工業大学情報工学部知能情報工学科卒業．平成 21 年大阪大学大学院情報科学研究科博士後期課程修了．同年日本学術振興会特別研究員（PD）．平成 22 年奈良先端科学技術大学院大学情報科学研究科助教．平成 26 年名古屋大学大学院情報科学研究科附属組込みシステム研究センター准教授．平成 29 年より同大学大学院情報学研究科附属組込みシステム研究センター准教授（改組による）．博士（情報科学）．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．

藤原 賢二 （正会員）

平成 22 年大阪府立工業高等専門学校総合工学システム専攻修了．平成 27 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年同大学院博士研究員．平成 28 年豊田工業高等専門学校情報工学科助教．博士（工学）．リファクタリングの適用履歴分析，プログラミング教育支援に関する研究に従事．



川島 尚己

平成 26 年京都工芸繊維大学工芸科学部情報工学課程卒業．平成 28 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．同年株式会社日立ソリューションズ入社．修士（工学）．ソフトウェア工学，特にリポジトリマイニングに興味を持つ．

飯田 元 （正会員）

昭和 63 年大阪大学基礎工学部情報工学科卒業．平成 3 年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．平成 7 年奈良先端科学技術大学院大学情報科学センター助教授．平成 17 年同大学情報科学研究科教授．博士（工学）．